GNFA $\rightarrow$ Regular Expression

Friday March 1, 2024
Announcements

• HW 3 out
  • Due Mon 3/4 12pm EST (noon)
Regular Expressions = Regular Langs?

*R* is a *regular expression* if *R* is:
1. *a* for some *a* in the alphabet *Σ*,
2. *ε*,
3. ∅,
4. \((R_1 \cup R_2)\), where *R_1* and *R_2* are regular expressions,
5. \((R_1 \circ R_2)\), where *R_1* and *R_2* are regular expressions, or
6. \((R_1^*)\), where *R_1* is a regular expression.

We would like it if:
- A *regular expression* represents a *regular language*
- The *set of all regular expressions* represents the *set of all regular languages*

(But we have to prove it)
Previously

**Thm:** A Lang is Regular \(\text{iff}\) Some Reg Expr Describes It

\[\Rightarrow\text{ If a language is regular, then it’s described by a reg expression}\]

\[\Leftarrow\text{ If a language is described by a reg expression, then it’s regular}\]

(Easier)

- **Key step:** convert reg expr \(\Rightarrow\) equivalent NFA!
- (Hint: we mostly did this already when discussing closed ops)

- **How to show that a language is regular?**
- **Construct a DFA or NFA!**
A regular expression is

1. a for some a in the alphabet Σ,
2. ε,
3. ∅,
4. \( R_1 \cup R_2 \), where \( R_1 \) and \( R_2 \) are regular expressions,
5. \( R_1 \circ R_2 \), where \( R_1 \) and \( R_2 \) are regular expressions,
6. \( R_1^* \), where \( R_1 \) is a regular expression.
Thm: A Lang is Regular iff Some Reg Expr Describes It

⇒ If a language is regular, then it’s described by a reg expression (Harder)
  • Key step: Convert an DFA or NFA → equivalent Regular Expression
  • First, we need another kind of finite automata: a GNFA

⇐ If a language is described by a reg expression, then it’s regular (Easier)
  • Key step: Convert the regular expression → an equivalent NFA!
  (full proof requires writing Statements and Justifications, and creating an “Equivalence” Table)
Generalized NFAs (GNFAs)

- GNFA = NFA with regular expression transitions

Transition can read multiple chars

plain NFA = GNFA with single char regular expr transitions

Goal: convert GNFAs to equivalent Regular Exprs
GNFA→RegExp function

On GNFA input $G$:
• If $G$ has 2 states, return the regular expression (on the transition), e.g.:

$$q_i \xrightarrow{(R_1)(R_2)^*(R_3) \cup (R_4)} q_j$$

Could there be less than 2 states?
GNFA to RegExp Preprocessing

- Modify input machine to have:
  - **New start state:**
    - No incoming transitions
    - $\varepsilon$ transition to old start state
  - **New, single accept state:**
    - With $\varepsilon$ transitions from old accept states

Does this change the language of the machine? i.e., are before/after machines equivalent?

Modified machine always has 2+ states:
- New start state
- New accept state
GNFA\rightarrow \text{RegExp} \text{ function (recursive)}

On GNFA input $G$:
- If $G$ has 2 states, return the regular expression (from transition), e.g.:

\[
\begin{array}{c}
q_i \\
\rightarrow \\
(R_1) (R_2)^* (R_3) \cup (R_4) \\
\rightarrow \\
q_j
\end{array}
\]

- Else:
  - “Rip out” one state
  - “Repair” the machine to get an equivalent GNFA $G'$
  - Recursively call GNFA\rightarrow \text{RegExp}(G')

Recursive definitions have:
- base case and
- recursive case
  (with “smaller” self-reference)
**GNFA→RegExp: “Rip/Repair” step**

To convert a GNFA -> regular expression:
1. “rip out” one state
2. “repair” machine to preserve equivalence,
3. repeat until only 2 states remain
**GNFA $\rightarrow$ RegExp:** “Rip/Repair” step

Before: two paths from $q_i$ to $q_j$:
1. Not through $q_{\text{rip}}$
2. Through $q_{\text{rip}}$

After:

To convert a GNFA $\rightarrow$ regular expression:
1. “Rip out” one state
2. “Repair” machine to preserve equivalence,
3. Repeat until only 2 states remain
GNFA $\rightarrow$ RegExpr: “Rip/Repair” step

Before:
- $q_i$ to $q_j$ through $R_4$
- $q_{rip}$ through $R_1$, $R_2$, $R_3$

After:
- Union of two “paths” from $q_i$ to $q_j$
  1. Not through $q_{rip}$
  2. Through $q_{rip}$

Expression:

$$(R_1)(R_2)^* (R_3) \cup (R_4)$$

To convert a GNFA $\rightarrow$ regular expression:
1. “Rip out” one state
2. “Repair” machine to preserve equivalence,
3. Repeat until only 2 states remain
GNFA → RegExpr: “Rip/Repair” step

Before:
- path through $q_{\text{rip}}$ has 3 transitions
- One is self-loop

After:
$$(R_1) (R_2)^* (R_3) \cup (R_4)$$
GNFA $\rightarrow$ RegExpr: “Rip/Repair” step

After:
- Self loop becomes star operation
- Others are concat’ed together

Before:
- path through $q_{rip}$ has 3 transitions
- One is self-loop
**Thm:** A language is regular iff some regular expression describes it.

⇒ If a language is regular, then it’s described by a regular expression.
   Need to convert DFA or NFA to regular expression...
   • Use GNFA→RegExpr to convert GNFA → equivalent regular expression!

⇔ If a language is described by a regular expression, then it’s regular.
   • Convert regular expression → equivalent NFA!

This time, let’s really prove equivalence!
(we previously “proved” it with some examples)
GNFA→RegExp function (recursive)

On GNFA input $G$:

- If $G$ has 2 states, return the regular expression (from transition), e.g.:

\[ q_i \xrightarrow{(R_1) (R_2)^* (R_3) \cup (R_4)} q_j \]

This time, let’s really prove equivalence! (we previously “proved” it with some examples)

- Else:
  - “Rip out” one state
  - “Repair” the machine to get an equivalent GNFA $G'$
  - Recursively call GNFA→RegExp($G'$)
GNFA $\rightarrow$ RegExpr: Rip/Repair Correctness

Must show these are equivalent
Equivalent = same language = accepts the same strings
**GNFA\(\rightarrow\)RegExpr: Rip/Repair Correctness**

Must show these are equivalent

\[(R_1) (R_2)^* (R_3) \cup (R_4)\]

**Must prove:**
- Every string accepted \textit{before} is accepted \textit{after}
- 2 cases:
  1. Let \(w_1 = \text{str accepted before}\), \(w_1\) doesn't go through \(q_{\text{rip}}\)
    - \(w_1\) still accepted after both use \(R_4\) transition
  2. Let \(w_2 = \text{str accepted before}\), \(w_2\) goes through \(q_{\text{rip}}\)
    - \(w_2\) accepted by \textit{after}?
    - Yes, via our previous reasoning
GNFA→RegExp function (recursive)

On GNFA input $G$:

- If $G$ has 2 states, return the regular expression (from transition), e.g.:

  \[ q_i \xrightarrow{(R_1)(R_2)^* (R_3) \cup (R_4)} q_j \]

- Else:
  - “Rip out” one state
  - “Repair” the machine to get an equivalent GNFA $G'$
  - Recursively call GNFA→RegExp($G'$)

Now we prove the whole function preserves equivalence

This time, let’s really prove equivalence! (we previously “proved” it with some examples)
**GNFA→RegExp Equivalence**

- **Equivalent** = the language does not change (same strings)!

Statement to Prove:  

\[ \text{LANGOF} (G) = \text{LANGOF} (R) \]

- where:  
  - \( G \) = a GNFA  
  - \( R \) = a Regular Expression = \( \text{GNFA→RegExp}(G) \)

Language could be infinite set of strings!

(how can we guarantee equivalence for a possibly infinite set of strings?)

This time, let’s **really prove** equivalence! (we previously “proved” it with some examples)

Recursion!
Inductive Proofs
(Proofs involving recursion)
Kinds of Mathematical Proof

• **Deductive proof** (from before)
  • **Start** with: assumptions, axioms, and definitions
  • **Prove**: news conclusions by making logical inferences (e.g., modus ponens)

• **Proof by induction** (i.e., “a proof involving recursion”) (now)
  • Same as above ...
  • But: use this when proving something that is **recursively** defined

A **valid recursive definition** has:
- base case(s) and
- recursive case(s) (with “smaller” self-reference)
Proof by Induction

To **Prove**: a *Statement* about a recursively defined “thing” \( x \):

1. **Prove**: *Statement* for **base case** of \( x \)

2. **Prove**: *Statement* for **recursive case** of \( x \):
   - **Assume**: induction hypothesis (IH)
     - *i.e.*, *Statement* is true for some \( x_{\text{smaller}} \)
     - *E.g.*, if \( x \) is number, then “smaller” = lesser number
   - **Prove**: *Statement* for \( x_{\text{larger}} \), using IH (and known definitions, theorems ...)
     - Typically: show that going from \( x_{\text{smaller}} \) to \( x_{\text{larger}} \) preserves *Statement*

A valid recursive definition has:
- base case(s) and
- recursive case(s) (with “smaller” self-reference)
Natural Numbers Are Recursively Defined

A Natural Number is:

- \( 0 \)
- \( k + 1 \), where \( k \) is a Natural Number

But definition is valid because self-reference is “smaller”

So proving things about Natural Numbers requires recursion in the proof, i.e., **proof by induction!**

A valid recursive definition has:
- **base case** and
- **recursive case** (with “smaller” self-reference)
Proof By Induction Example (Sipser Ch 0)

Prove true: \( P_t = PM^t - Y \left( \frac{M^t - 1}{M - 1} \right) \)

- \( P_t \) = loan balance after \( t \) months
- \( t \) = # months
- \( P \) = principal = original amount of loan
- \( M \) = interest (multiplier)
- \( Y \) = monthly payment

(Details of these variables not too important here)
Proof By Induction Example (Sipser Ch 0)

Prove true: \( P_t = P \cdot M^t - Y \left( \frac{M^t - 1}{M - 1} \right) \)

Proof: by induction on natural number \( t \)

Base Case, \( t = 0 \):

- Goal: Show \( P_0 = P \) (amount owed at start = loan amount)
- Proof of Goal:
  \[
  P_0 = P \cdot M^0 - Y \left( \frac{M^0 - 1}{M - 1} \right) = P
  \]
  
  - Plug in \( t = 0 \)
  - Simplify, to get to goal statement

An proof by induction exactly follows the recursive definition (here, natural numbers) that the induction is “on”

A Natural Number is:
- 0
- Or \( k + 1 \), where \( k \) is a natural number
**Proof By Induction Example (Sipser Ch 0)**

Prove true: \( P_t = PM^t - Y \left( \frac{M^t - 1}{M - 1} \right) \)

**Inductive Case:** \( t = k + 1 \), for some nat num \( k \)

- Inductive Hypothesis (IH), assume statement true for some \( t = (\text{smaller}) k \):
  \[
  P_k = PM^k - Y \left( \frac{M^k - 1}{M - 1} \right)
  \]

- Goal statement to prove, for \( t = k + 1 \):
  \[
  P_{k+1} = PM^{k+1} - Y \left( \frac{M^{k+1} - 1}{M - 1} \right)
  \]

- Proof of Goal:
  \[
  P_{k+1} = P_k M - Y
  \]

**A proof by induction** exactly follows the recursive definition (here, natural numbers) that the induction is “on”

A Natural Number is:
- \( 0 \)
- \( k + 1 \), for some nat num \( k \)

“Connect together” known definitions and statements

Plug in IH for \( P_k \)

Simplify, to get to goal statement

**Definition of Loan:**
amt owed in month \( k+1 \) = amt owed in month \( k \) * interest \( M \) – amt paid \( Y \)
In-class Exercise: Proof By Induction

Prove: \( (z \neq 1) \)

\[
\sum_{i=0}^{m} z^i = \frac{1 - z^{m+1}}{1 - z}
\]

Use **Proof by Induction**.

Make sure to clearly state what (number) the induction is “on”
Proof by Induction: CS 622 Example

**Statement to prove:** \( \text{LANGOF}(G) = \text{LANGOF}(R = \text{GNFA}\rightarrow \text{RegExpr}(G)) \)

- **Where:**
  - \( G = \) a GNFA
  - \( R = \) a Regular Expression
  - \( R = \text{GNFA}\rightarrow \text{RegExpr}(G) \)

- i.e., \( \text{GNFA}\rightarrow \text{RegExpr} \) must not change the language!
  - Key step: the rip/repair step

**Condition for \( \text{GNFA}\rightarrow \text{RegExpr} \) function to be “correct”, i.e., the languages must be equivalent**

Now we are **really** proving equivalence! (previously, we “proved” equivalence with a table of examples)
Last Time: **GNFA→RegExpr** (recursive) function

On GNFA input $G$:

- If $G$ has 2 states, **return the regular expression** (from the transition), e.g.: $q_i \xrightarrow{(R_1)(R_2)^*(R_3) \cup (R_4)} q_j$

- Else:
  - “Rip out” one state
  - “Repair” the machine to get an equivalent GNFA $G'$
  - **Recursively call GNFA→RegExpr($G'$)**

Recursive definitions have:
- base case and
- recursive case (with a “smaller” object)
Proof by Induction: CS 622 Example

Statement to prove: \( \text{LANGOF}(G) = \text{LANGOF}(\text{GNFA} \Rightarrow \text{RegExpr}(G)) \)

Proof: by Induction on # of states in \( G \)

1. Prove Statement is true for base case

\( G \) has 2 states

\begin{tikzpicture}
  \node (q_i) at (0,0) [circle,draw] {\( q_i \)};
  \node (q_j) at (1,0) [circle,draw] {\( q_j \)};
  \draw [->] (q_i) edge [loop above] node [above] {\( R \)} (q_i);
  \draw [->] (q_i) edge node [right] {\( R \)} (q_j);
\end{tikzpicture}

Don’t forget to write out Statements / Justifications!

States:
1. \( \text{LANGOF}(\begin{array}{c}
    q_i \\
    \text{R} \\
    q_j
  \end{array}) = \text{LANGOF}(R) \)
2. \( \text{GNFA} \Rightarrow \text{RegExpr}(\begin{array}{c}
    q_i \\
    \text{R} \\
    q_j
  \end{array}) = R \)

Justifications:
1. Definition of GNFA
2. Definition of GNFA \( \Rightarrow \) RegExpr
3. From (1) and (2)
Proof by Induction: CS 622 Example

Statement to prove: \( \text{LANGOF}(G) = \text{LANGOF}(\text{GNFA} \rightarrow \text{RegExpr}(G)) \)

Proof: by Induction on # of states in \( G \)

1. **Prove Statement is true for base case**
   - \( G \) has 2 states

2. **Prove Statement is true for recursive case:**
   - **Assume** the induction hypothesis (IH):
     - **Statement** is true for smaller \( G' \)
   - **Use** it to prove **Statement** is true for larger \( G \)
     - Show that going from \( G \) to \( G' \) preserves **Statement**

Don’t forget to write out Statements / Justifications!

Show that “rip/repair” step converts \( G \) to smaller, equivalent \( G' \)
Proof by Induction: CS 622 Example

Statement to prove: \( \text{LANGOF}(G) = \text{LANGOF}(\text{GNFA}\rightarrow\text{RegExpr}(G)) \)

Proof: by Induction on # of states in \( G \)

1. \( \text{Prove Statement} \) is true for base case \( G \) has 2 states

   \( q_i \xrightarrow{R} q_j \)

2. \( \text{Prove Statement} \) is true for recursive case: \( G \) has > 2 states

   \( \text{LANGOF}(G') = \text{LANGOF}(\text{GNFA}\rightarrow\text{RegExpr}(G')) \)
   (Where \( G' \) has less states than \( G \))

   \( \text{LANGOF}(G) = \text{LANGOF}(\text{GNFA}\rightarrow\text{RegExpr}(G')) \)

   \( \text{LANGOF}(G) = \text{LANGOF}(\text{GNFA}\rightarrow\text{RegExpr}(G)) \)

Statements
1. \( \text{LANGOF}(G') = \text{LANGOF}(\text{GNFA}\rightarrow\text{RegExpr}(G')) \)
2. \( \text{LANGOF}(G) = \text{LANGOF}(G') \)
3. \( \text{GNFA}\rightarrow\text{RegExpr}(G) = \text{GNFA}\rightarrow\text{RegExpr}(G') \)
4. \( \text{LANGOF}(G) = \text{LANGOF}(\text{GNFA}\rightarrow\text{RegExpr}(G)) \)

Justifications
1. IH
2. Correctness of Rip/Repair step (prev)
3. Def of GNFA\rightarrow\text{RegExpr}
4. From (1), (2), and (3)
Thm: A Lang is Regular \textit{iff} Some Reg Expr Describes It

\[ \Rightarrow \text{If a language is regular, it is described by a regular expr} \]

Need to convert DFA or NFA to Regular Expression ...

✓ • Use GNFA$\Rightarrow$RegExpr to convert GNFA $\rightarrow$ equiv regular expression!

\[ \Leftarrow \text{If a language is described by a regular expr, it is regular} \]

✓ • Convert regular expression $\rightarrow$ equiv NFA!

Now we may use regular expressions to represent regular langs.

So a regular language has these equivalent representations:
- DFA
- NFA
- Regular Expression

So we also have another way to prove things about regular languages!
So Far: How to Prove A Language Is Regular?

Key step, either:

• Construct DFA

• Construct NFA

• Create Regular Expression

Slightly different because of recursive definition

$R$ is a regular expression if $R$ is

1. $a$ for some $a$ in the alphabet $\Sigma$,
2. $\varepsilon$,
3. $\emptyset$,
4. $(R_1 \cup R_2)$, where $R_1$ and $R_2$ are regular expressions,
5. $(R_1 \circ R_2)$, where $R_1$ and $R_2$ are regular expressions, or
6. $(R_1^*)$, where $R_1$ is a regular expression.